# Анотація

Курсова робота присвячена вибраним аспектам методології розробки програмного забезпечення – ООП, дослідженню, розробці та побудові на мові C# ієрархічної моделі класів, що відображає реальну модель реальних об’єктів. Проаналізовано конкретну задачу для вибору взаємозв’язків між елементами моделі.

Обґрунтовано вибір тієї чи іншої інструментальної можливості мови програмування для реалізації ієрархічної моделі класів, розробки класу та їх взаємозв’язків.

The course work is devoted to selected aspects of software development methodology - the OOP, research, development and construction of the hierarchical model of classes on C# programming language, which reflects the real model of real objects. Also the specific task to select relationships between the elements of the model was analyzed.

There was described the choice of instrumental possibilities of the programming language to implement hierarchical model of classes, develop classes and their relationships.

**зміст**

[Анотація 4](#_Toc451288586)

[Вступ 7](#_Toc451288587)

[Основна частина 9](#_Toc451288588)

[1. Вхідні дані 9](#_Toc451288589)

[2. Ієрархічна модель класів 10](#_Toc451288590)

[2.1 Діаграма класів 10](#_Toc451288591)

[2.2 Обґрунтування вибору взаємозв’язків ієрархії 11](#_Toc451288592)

[3. Опис побудованих класів 13](#_Toc451288593)

[3.1 Спільні аспекти реалізації класів 13](#_Toc451288594)

[3.2 Class Date 14](#_Toc451288595)

[3.3 Class Goods та enum GoodsType 16](#_Toc451288596)

[3.4 Class Seller 18](#_Toc451288597)

[3.5 Class Sale 19](#_Toc451288598)

[3.6 Class Balance 21](#_Toc451288599)

[3.7 Реалізація патерну Observer 25](#_Toc451288600)

[Висновки 26](#_Toc451288601)

[Список використаних джерел 28](#_Toc451288602)

[Додаток 1 29](#_Toc451288603)

[1. Class Date 29](#_Toc451288604)

[2. Class Seller 32](#_Toc451288605)

[3. Class Goods 34](#_Toc451288606)

[4. Class Sale 37](#_Toc451288607)

[5. Class Balance 40](#_Toc451288608)

[6. Interface ITable 45](#_Toc451288609)

[Додаток 2 46](#_Toc451288610)

[1. Опис програми 46](#_Toc451288611)

[2. Робота програми 46](#_Toc451288612)

# Вступ

Автоматизація бізнес систем, технологічних процесів та інших сфер життєдіяльності людини сьогодні є однією з найважливіших технічних задач. Кожний програмний продукт має своє цільове застосування, відповідно до якого він розроблюється. Процес розробки є дуже складним і, для полегшення розробки і підтримки продукту, структурованим на етапи.

Основними етапами розробки є:

1. Аналіз вимог для програмного забезпечення. Тобто, збір необхідних даних досліджуваної області та ратифікація вимог.
2. Проектування програмного забезпечення: розробка графічної моделі Уніфікованою мовою моделювання (UML) і стандарти формату описів.
3. Інженерія програмного забезпечення: безпосереднє кодування моделі.
4. Тестування програмного забезпечення: відлагодження та виявлення помилок розробки.
5. Підтримка програмного забезпечення: програмні системи часто мають проблеми сумісності та переносимості, а також потребують подальших модифікацій протягом довгого часу після того, як створена їх перша версія.
6. Керування конфігурацією програмного забезпечення: оскільки системи програмного забезпечення дуже складні та модифікуються в процесі експлуатації, їх конфігурації повинні управлятися стандартизованим та структурованим методом.

Побудову ієрархічної моделі класів не можна чітко віднести до певного етапу тому, що процес є складовою частиною як проектування так і інженерії програмного забезпечення, проте можна чітко окреслити важливість даного процесу.

Для цього приведемо порівняння з автомобільною платформою, що є своєрідним аналогом в автомобільному виробництві. Платформа – це набір базових компонентів, використовуючи які будується автомобіль та подальші його модифікації. Зазвичай, на одній і тій самій платформі автомобілі розроблюються впродовж багатьох років, тому що вона є фундаментальним компонентом і її зміна надто дороге задоволення. З цього можна зробити висновки про важливість даного процесу і сформувати мету даною роботи.

Розумно розроблена і правильно запрограмована модель присікає помилки процесу розробки на ранньому етапі та закладає наріжний камінь до легкої підтримки і подальшої модифікації системи.

Метою курсової роботи є набуття навиків побудови ієрархії класів та створення програмного продукту.

Завданням курсової роботи є створення на мові C# ієрархічну модель класів, що відображає реальну модель реальних об’єктів та на її основі створити функціональний приклад системи.

# Основна частина

## Вхідні дані

Для курсової роботи було запропоновано мову програмування C# та наступна структура об’єкту реального світу для моделювання ієрархії класів (Таблиця 1.1. Структура об’єкту реального світу):

Таблиця .. Структура об’єкту реального світу

|  |  |
| --- | --- |
| Дата | Місяць;  День. |
| Товар | Назва;  Дата (надходження);  Тип (Ваговий, розпакований);  Ціна. |
| Продавець | Прізвище;  Ім’я;  Дата (прийняття на роботу). |
| Операція Продажу | Дата (продажу);  Продавець;  Товар;  Кількість товару. |
| Денний баланс | Список операцій |

## Ієрархічна модель класів

### Діаграма класів

Для графічного представлення за допомогою мови моделювання UML (Unified Modeling Language) була створена діаграма взаємозв’язків класів (Рисунок 2.1 Діаграма ієрархії класів).



Рисунок . Діаграма ієрархії класів

### Обґрунтування вибору взаємозв’язків ієрархії

Для побудови ієрархії моделі існує три типи взаємозв’язків: агрегація, композиція та пряме наслідування.

**Агрегація** – тип взаємозв’язку, коли об’єкт одного класу є частиною об’єкту іншого класу, при чому об’єкт, що є компонентом не може існувати без цілого. Тобто, ціле жорстко регламентує час життя своєї компоненти.

**Композиція** – тип взаємозв’язку, коли об’єкт одного класу є частиною об’єкту іншого класу, при чому об’єкт, що є компонентом може існувати самостійно.

**Наслідування** – тип взаємозв’язку, коли між все, що справедливо для базового класу – справедливо і для похідного. Тобто, умовно можна сказати «є», наприклад, студент є людиною.

В розробленій відповідно до завдання ієрархії класів відсутнє пряме наслідування. Це пояснюється тим, що базові класи є лише частинами похідного класу.

Класи Seller, Goods та Sale мають як поле об’єкт типу Date. В усіх трьох випадках використовується тип зв’язку агрегація , тому що дата є лише компонентом класів операції продажу, товару та продавця. Таким чином при розробці більш складної системи на основі цієї ієрархії, в будь-яких операціях проведених в одну дату, можна встановити посилання лише на один об’єкт, що значно заощадить місце в пам’яті. Також, якщо буде необхідність порівнювати об’єкти на відповідність, потрібно буде перевірити лише посилання, а не всі поля об’єктів, що значно зменшить час виконання операції.

В класі Sale, окрім поля типу Date, є поля типу Seller та Goods. Обидва елементи є складовими частинами типу Sale. В випадку класу Seller використовується агрегація, тому що немає сенсу кожного разу створювати нового продавця в системі при кожній операції продажу. Набагато розумнішим, на мою думку, варіантом є встановлення посилань на продавців (об’єкти типу Seller) в операції продажу (об’єкт типу Sale). Попри аналогічні переваги, які описані для об’єктів типу Date, у випадку, якщо продавець змінить прізвище чи ім’я, його потрібно буде змінити лише для одного об’єкту, а не для всіх.

До класу Goods відносно Sale використовується композиція. Композицію було вибрано через те, що в класі Goods є поле типу double що відповідає за ціну товару. Ціна товару може змінюватись в залежності від різних обставин для одного і того ж товару. Тому, схема використання один товар – безліч посилань, в даному випадку є помилкою.

Між класами Balance та Sale встановлена агрегація. Виходячи з вхідних даних однозначно визначити який тип зв’язку, агрегацію чи композицію, використовувати не можливо. Можна лише стверджувати, що пряме наслідування є невірним варіантом відношення.

## Опис побудованих класів

### Спільні аспекти реалізації класів

Усі класи реалізують інтерфейс ISpreadsheet і визначений в ньому метод string ToTableRow(), що призначений для виводу об’єкту класу на екран у вигляді рядка таблиці. Похідні класи в своїй реалізації викликають уже реалізовані методи своїх базових класів та доповнюють своїми полями в рядковому вигляді.

Також кожен клас має перевизначений метод bool Equals(object obj) та string ToString(). Перший використовується для порівняння об’єктів на ідентичність. Застосовується в кінцевому похідному класі для запобігання додавання в список операцій повторень, де для перевірки повторень використовується метод колекції List<T>.Contains(object obj). В свої реалізації він використовує метод bool Equals(object obj), який є у будь-яких об’єктів в мові C#.

Метод string ToString() переводить об’єкт в рядок, що дає змогу зручно записувати об’єкт у текстовий файл.

### Class Date

Клас розроблений для роботи з датою (число, місяць) (Рисунок 3.1).



Рисунок . Графічний вигляд класу Date

#### Вибір конкретних типів властивостей для полів

Автоматичні властивості не підходили для реалізації тому, що було необхідно було накласти додаткову логіку перевірки значення коректного значення на метод - мутатор . Тобто перевірка на місяць [1;12] та день місяця [1; MaxDayMonth(int month)]. MaxDayMonth(int month) – статична функція, що в залежності від аргументу повертає максимальну кількість днів в місяці. Оголошена модифікатором static, тому що реалізація не належить до якогось певного об’єкту, а має загальне призначення.

Усі похідні класи від класу Date не потребують можливості змінювати дату після її ініціалізації, тому обидва методи set помічені модифікатором доступу protected. Через це об’єкт класу можна ініціалізувати виключно через конструктор.

Мутатори методів, при спробі присвоїти полям некоректні дані, генерують ArgumentOutOfRangeException() з повідомленням про помилку. Вибір на користь виключних ситуацій зроблений через те, що кінцевим користувачем класів може бути будь-хто і при спробі присвоїти хибні значення має отримати відповідну реакцію на свої дії.

#### Вибір певних перевантажених операцій для та їх зміст

Було реалізовано перевантаження операцій порівняння «>» та «<», які використовуються в кінцевому класі в методі void Add(Sale obj), який відповідає за додавання операції продажу в хронологічному порядку.

### Class Goods та enum GoodsType

Клас розроблений для роботи з товаром (Найменування, ціна, дата надходження, тип (Ваговий, розпакований)). Для типу товару розроблено перерахування GoodsType (Рисунок 3.2).



Рисунок . Графічний вигляд класу Goods

#### Вибір конкретних типів властивостей для полів

Усі методи доступу set властивостей помічені модифікатором protected, окрім Type. GoodsType Type - автоматична властивість, тому що вона не потребує додаткової логіки, але реалізує універсальний інтерфейс класу.

Усі властивості, окрім GoodsType, є звичайними властивостями.

Властивість double Price в методі доступу set має діапазон допустимих значень (0;∞], якщо значення виходить за діапазон генерується ArgumentOutOfRangeException().

Властивість string Name методу доступу set відповідає за правильну ініціалізацію найменування товару. Тому вона перевіряє вхідний аргумент на відповідність null або String.Empty. Якщо умова виконується, то генерується ArgumentNullException().

Date Receiving методу доступу set відповідає за правильну ініціалізацію дати надходження товару. В тілі set методу виконується перевірка чи об’єкт-аргумент рівний null. Якщо умова виконується, то генерується ArgumentNullException().

### Class Seller

Клас розроблений для роботи з продавцем (Прізвище, ім’я, дата (прийняття на роботу)) (Рисунок 3.3).



Рисунок . Графічний вигляд класу Seller

#### Вибір конкретних типів властивостей для полів

Властивість string Name та string Name методу доступу set відповідають за правильну ініціалізацію ім’я та прізвища продавця. Тому вона перевіряє вхідний аргумент на відповідність null або String.Empty. Якщо умова виконується, то генерується ArgumentNullException(). До властивості реалізований відкритий доступ public get та set методів доступу, тому що в силу тих чи інших обставин прізвище та ім’я продавця може змінюватись.

Властивість Date Recruited методу доступу set помічена модифікатором protected відповідає за правильну ініціалізацію дати отримання роботи. В тілі set методу виконується перевірка чи об’єкт-аргумент рівний null. Якщо умова виконується, то генерується ArgumentNullException().

### Class Sale

Клас розроблений для роботи з операцією продажу (Дата (продажу), продавець, товар, кількість товару) (Рисунок 3.4).



Рисунок . Графічний вигляд класу Sale

#### Вибір конкретних типів властивостей для полів

Усі методи доступу set властивостей помічені модифікатором protected, для того, щоб об’єкт класу неможливо було змінити після ініціалізації.

У властивості double Amount методу set реалізована перевірка на допустимий діапазон (0, ∞], якщо аргумент виходить за діапазон генерується ArgumentOutOfRangeException().

Date SaleDate та Seller SellerTransaction організовані по одному принципу: метод-мутатор перевіряє чи аргумент рівний null, якщо так, то генерує ArgumentNullException(). Аксесор повертає посилання на об’єкти.

Для безпеки збереження інформації об’єкту метод get властивості GoodTransaction повертає копію на поле goodsTransaction. Set метод перевіряє чи аргумент рівний null, якщо так, то генерує ArgumentNullException(), якщо ні, то копіює вміст аргументу за допомогою конструктора копіювання.

Для підрахунку ціни проданого товару створено read-only властивість double TotalPrice, яка повертає ціну обраховану за формулою (Формула 3.1):

(Формула .)

### Class Balance

Клас розроблений для роботи з списком операцій продажу (Список операцій) (Рисунок 3.5), реалізує інтерфейси IObservable та (Рисунок 3.6).



Рисунок . Графічне зображення класу Balance

#### Вибір певних фрагментів програмного коду, для яких встановлена обробка виключних ситуацій

Для всіх функцій запису та зчитування з файлу було встановлено обробку виключних ситуацій. Помилка може трапитись як під час запису так і під час зчитування. При записі в файл, такий файл може вже бути відкритий в іншій програмі чи може бути статись помилка файлової системи. При зчитуванні – дані можуть бути, наприклад, пошкоджені. В таких ситуаціях зазвичай генерується виключна ситуація. Для відновлення роботи програми, небезпечний код було заключено в блоки try{}finally{}. Оброблення виключної ситуації в цих методах відсутнє, їх має забезпечити метод, що викликає функції запису/зчитування. При зчитуванні чи запису в файл обов’язково спочатку треба відкрити потік вводу/виводу. Якщо згенерується виключна ситуація потік все одно залишиться відкритим. Тому в блоці finally{}, який виконається незалежно від того чи виключна ситуація згенерується чи ні організовано закриття файлового потоку.

#### Вибір типу делегата для обчислення статистичних функцій.

Для обчислення статистичних функцій був вибраний тип делегата public delegate Sale Statistics(). Статистичними функціями в класі представлені пошук максимальної та мінімальної покупки public Sale Max() та public Sale Min(). Делегат дозволяє передати в функцію частину коду, який описує за яким критерієм буде вибиратись елемент для мінімальної чи максимальної покупки.

#### Пошук за числовим та текстовим полем.

Для пошуку була створена функція public Balance Search(Compare del, object obj) та оголошений відповідний делегат public delegate bool Compare(Sale, object obj),який визначає критерій за яким буде здійснюватися пошук. Відповідно реалізовані функції пошуку за текстовим полем public bool CompareByName(Sale sale, object name) та за числовим полем public bool CompareByPrice(Sale sale, object price). При пошуку об’єкта в функцію залежно від вибору передається відповідна функція. На виході маємо масив операцій, що задовольняють нашому критерію пошуку.

#### Індексатор

Індексатор дозволяє поводитись з об’єктом класу, ніби з масивом. Для зручності доступу за індексом до списку операцій List<Sale> було застосовано read-only індексатор public Sale this[int index]. При отриманні відповідного об’єкту index перевіряється на діапазон допустимих значень, якщо index виходить за діапазон, то посилається виключення IndexOutOfRangeException().

#### Запис даних у бінарний файл

Для запису даних в бінарний файл кожен клас позначений атрибутом [Serializable()], що дозволяє виконати операцію серіалізації (переводу класу в потік байтів) для всіх об’єктів даного класу.

Для запису в файл створюється чи відкривається файл з режимом FileMode.OpenOrCreate та доступом FileAccess.Write, що дозволяє безпечно записувати в існуючий файл не порушуючи даних або створити новий файл.

#### Зчитування даних у бінарний файл

Зчитування відбувається за рахунок механізму десеріалізації даних. Щоб ненароком не пошкодити структуру файлу використовується доступ тільки для читання FileMode.Open.

Метод, що відповідає за зчитування з файлу оголошено статичним, тому що він не прив’язаний до конкретного об’єкту та таким чином надає більшу гнучкість дій з зчитаним списком операцій.

#### Запис даних у текстовий файл

Для запису даних в текстовий файл створюється потік для запису StreamWriter і за допомогою метода StreamWriter.WriteLine(String line) проводиться запис. Для переведення об’єкту в текст використовується перевизначений метод string ToString().

#### Зчитування даних у бінарний файл

Для зчитування даних оголошена статична функція public static List<Sale> ReadFromTextFile(string fileName). Зчитування відбувається за рахунок StreamReader.ReadLine(), яка знаходиться в циклі, що виконується поки не настане кінець файлу (StreamReader.Peek() != -1)

### Реалізація патерну Observer

Для реалізації патерну було створено два інтерфейси IObservable та IObserver та оголошено тип делегату NotifyObserver (Рисунок 3.6).



Рисунок .

IObservable призначений для реалізації класу-джерела. IObserver - для реалізації класу-спостерігача. Даний патерн дозволяє відслідковувати зміни об’єкту джерела об’єктам спостерігачам. Тип делегату використовується як тип події.

# Висновки

Отже, в процесі виконання курсової роботи було досліджено головні аспекти процесу створення ієрархії моделі класів та виявлено проблемні моменти. Дослідження проводились на платформі .NET Framework та мові програмування C#. За допомогою цієї мови було розроблено програму на основі моделі реального світу.

Ключовими проблемними питаннями, на мою думку, являються зниження продуктивності програмного продукту за рахунок використання принципів ООП та заплутаність розгалуження при побудові складної ієрархії класів. При побудові великої ієрархії кінцевий клас не завжди має той вигляд, який передбачався спочатку. Наприклад, клас може отримати надлишковий функціонал, який не буде використовуватись, але буде знижувати продуктивність системи. На рахунок цього, чудовим але дещо трудомістким рішенням проблеми має мова програмування Erlang. При реалізації успадковування вона вимагала перерахувати разом з класом усі методи, що мають успадковуватись. Таким чином програміст отримував би лише необхідні методи. Тобто існує проблема надмірної універсальності.

Також є проблема надмірного поліморфізму. Часто виникає проблема надмірної логіки в розроблених класах. Через це їх важко розширювати. Тому необхідно правильно розуміти що з себе представлятиме клас ще на етапі проектування.

В процесі побудови ієрархічної моделі класів можна зробити висновок, що за типом зв’язку при можливості вибору кращим є агрегація. Вона дозволяє динамічно та гнучко керувати об’єктами успадковування.

В процесі розробки моделі класу було набуто навички побудови ієрархії моделі класів і розроблено відповідну програму з базовими функціями для демонстрації. Завдяки набутим навикам, модель була спроектована легко розширюваною без зайвого функціоналу

# Список використаних джерел

1. Шилдт Г. C# 4.0 Полное руководство – М.: Вильямс, 2011. – 304с.
2. Эндрю Троелсен. Язык Программирования C# 5.0 и платформа .NET 4.5 (6-е издание 2013). – 322с.
3. Бишоп Дж., Хорспул Н. C# в кратком изложении. – М.: Бином. Лаборатория знаний, 2005. – 472с.
4. Прайс Д., Гандэрлой М. Visual C# .NET. Полное руководство. – К.: ВЕК+, СПб.: КОРОНА принт, К.: НТИ, М.: Энтроп, 2004. – 960с.

http://cyberforum.ru

# Додаток 1

## Class Date

using System;

namespace CourseWork.Model

{

//Дата {Місяць, День};

[Serializable()]

class Date: ISpreadsheet

{

public const int MinDay = 1;

public const int MinMonth = 1;

public const int MaxMonth = 12;

private int day;

private int month = MinMonth; // 1 - 12, Jan - Dec

public Date(string day, string month)

: this(Int32.Parse(day), Int32.Parse(month))

{

}

public Date(int day, int month)

{

Day = day;

Month = month;

}

public Date(Date date)

{

this.day = date.day;

this.month = date.month;

}

public int Day

{

get

{

return day;

}

protected set

{

if (value <= MaxDayMonth(month) && value >= MinDay)

{ day = value; }

else

{

throw new ArgumentOutOfRangeException(SupportText.DateDay,

SupportText.ArgumentOutOfRange + ": " + MinDay + '-'

+ MaxDayMonth(month).ToString());

}

}

}

public int Month

{

get

{

return month;

}

protected set

{

if (value >= MinMonth && value <= MaxMonth)

{ month = value; }

else

{

throw new ArgumentOutOfRangeException(SupportText.DateMonth,

SupportText.ArgumentOutOfRange + ": " + MinMonth + '-'

+ MaxMonth);

}

}

}

public static int MaxDayMonth(int month)

{

if (month > MaxMonth || month < MinMonth)

{

throw new ArgumentOutOfRangeException(

SupportText.DateMonth,

SupportText.ArgumentOutOfRange);

}

int max;

switch (month)

{

case 4:

case 6:

case 9:

case 11: max = 30; break;

case 2: max = 28; break;

default: max = 31; break;

}

return max;

}

public static bool operator <(Date obj1, Date obj2)

{

if (obj1 == null)

return true;

else if (obj2 == null)

return false;

return (obj1.month < obj2.month || (obj1.month == obj2.month && obj1.day < obj2.day));

}

public static bool operator >(Date obj1, Date obj2)

{

if (obj1 == null)

return false;

else if (obj2 == null)

return true;

return (obj1.month > obj2.month || (obj1.month == obj2.month && obj1.day > obj2.day));

}

public string ToTableRow()

{

const string Format = "{0, 2:00}.{1, 2:00}\u2502";

return string.Format(Format, day, month);

}

public override string ToString()

{

return day + ' '.ToString() + month;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Date date = obj as Date;

if (date == null)

{ return false; }

return (date.day == day && Month == date.month);

}

}

}

## Class Seller

using System;

namespace CourseWork.Model

{

//Продавець {Прізвище, Ім’я, Дата (прийняття на роботу)};

[Serializable()]

class Seller: ISpreadsheet

{

private string name;

private string surname;

private Date recruited;

public Seller(string name, string surname, Date date)

{

Name = name;

Surname = surname;

recruited = new Date(date);

}

public Seller(string day, string month, string name, string surname)

{

Name = name;

Surname = surname;

recruited = new Date(day, month);

}

public string Name

{

get

{

return name;

}

set

{

if (String.IsNullOrEmpty(value))

{

throw new ArgumentNullException(SupportText.SellerName,

SupportText.ArgumentNull);

}

else

{

name = value;

}

}

}

public string Surname

{

get

{

return surname;

}

set

{

if (String.IsNullOrEmpty(value))

{

throw new ArgumentNullException(SupportText.SellerSurname,

SupportText.ArgumentNull);

}

else

{ surname = value; }

}

}

internal Date Recruited

{

get

{

return recruited;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Date,

SupportText.ArgumentNull);

recruited = value;

}

}

public override string ToString()

{

return recruited.ToString() + ' ' + Name + ' ' + Surname;

}

public new bool Equals(object obj)

{

if (obj == null)

return false;

Seller seller = obj as Seller;

if (seller == null)

{ return false; }

return (Name.Equals(seller.Name) && Surname.Equals(seller.Surname)

&& recruited.Equals(seller.recruited));

}

public string ToTableRow()

{

const string Format = "{0, 14}\u2502";

return string.Format(Format, Surname);

}

}

}

## Class Goods

using System;

namespace CourseWork.Model

{

//Товар {Назва, Дата (надходження), Тип (Ваговий, розпакований), Ціна}

/// <summary>

/// Перелічування стану товарів

/// </summary>

public enum GoodsType { Unpacked, WeighedOut }

[Serializable()]

class Goods : ISpreadsheet

{

private string name; // Ім'я товару

private double price; // Ціна товару

private Date receiving;

public Goods(string name, double price, Date date, GoodsType type)

{

Name = name;

Price = price;

Receiving = date;

Type = type;

}

public Goods(string day, string month, string name, string price, string type)

{

Name = name;

Price = Double.Parse(price);

Type = (GoodsType)Enum.Parse(typeof(GoodsType), type);

Receiving = new Date(day, month);

}

public Goods(Goods goods)

{

this.name = goods.name;

this.price = goods.price;

receiving = new Date(goods.Receiving);

}

public GoodsType Type

{

get; protected set;

}

public double Price

{

get

{ return price; }

protected set

{

if (value > 0.0)

{ price = value; }

else

{ throw new ArgumentOutOfRangeException(

SupportText.Price, SupportText.ArgumentOutOfRange

+ ": 0.0 <"); }

}

}

public string Name

{

get

{

return name;

}

protected set

{

if (String.IsNullOrEmpty(value)) // якщо null або ""

{

throw new ArgumentNullException(SupportText.GoodsName,

SupportText.ArgumentNull);

}

else

{ name = value; }

}

}

public Date Receiving

{

get

{

return receiving;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Date,

SupportText.ArgumentNull);

receiving = value;

}

}

public override string ToString()

{

return receiving.ToString() + ' ' + name + ' ' + price + ' ' + Enum.GetName(typeof(GoodsType), Type);

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Goods goods = obj as Goods;

if (goods == null)

return false;

return (receiving.Equals(goods.receiving) && price == goods.price

&& Name.Equals(goods.Name) && Type == goods.Type);

}

/// <summary>

public string ToTableRow()

{

const string Format = "{0, 14}\u2502{1, 11}\u2502{2,7:##0.0}\u2502";

return String.Format(Format, name, (Type == GoodsType.Unpacked)

? SupportText.GoodsTypeUnpacked

: SupportText.GoodsTypeWeighedOut, price);

}

}

}

## Class Sale

using System;

namespace CourseWork.Model

{

//Операція Продажу { Дата (продажу), Продавець, Товар, Кількість товару };

[Serializable()]

class Sale : ISpreadsheet

{

private double amount = 1; // кількість товару

private Date saleDate;

private Goods goodsTransaction;

private Seller seller;

public Sale(Seller seller, Goods goods, Date saleDate, double amount)

{

SellerTransaction = seller;

GoodsTransaction = goods;

SaleDate = saleDate;

Amount = amount;

}

public Sale(string saleDay, string saleMonth, string sellerEmployDay,

string sellerEmployMonth, string sellerName, string sellerSurname,

string goodsReceiptDay, string goodsReceiptMonth,

string goodsName, string goodsPrice, string goodsType,

string amount)

{

SellerTransaction = new Seller(sellerEmployDay, sellerEmployMonth,

sellerName, sellerSurname);

goodsTransaction = new Goods(goodsReceiptDay, goodsReceiptMonth,

goodsName, goodsPrice, goodsType);

SaleDate = new Date(saleDay, saleMonth);

Amount = Double.Parse(amount);

}

internal Seller SellerTransaction

{

get

{

return seller;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Goods,

SupportText.ArgumentNull);

seller = value;

}

}

internal Goods GoodsTransaction

{

get

{

return goodsTransaction;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Goods,

SupportText.ArgumentNull);

goodsTransaction = new Goods(value);

}

}

internal Date SaleDate

{

get

{

return saleDate;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Date,

SupportText.ArgumentNull);

saleDate = value;

}

}

public double Amount

{

get

{

return amount;

}

set

{

if (value > 0.0)

{ amount = value; }

else

{

throw new ArgumentOutOfRangeException(

SupportText.SaleAmount, SupportText.ArgumentOutOfRange

+ ": 0.0 <");

}

}

}

public double TotalPrice

{

get

{

return amount \* GoodsTransaction.Price;

}

}

public override string ToString()

{

return SaleDate.ToString() + ' ' + SellerTransaction.ToString() + ' ' + GoodsTransaction.ToString() + ' ' + Amount;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Sale sale = obj as Sale;

if (sale == null)

return false;

if (sale == this)

{

return true;

}

return (amount == sale.amount && saleDate.Equals(sale.saleDate)

&& SellerTransaction.Equals(sale.SellerTransaction)

&& GoodsTransaction.Equals(sale.GoodsTransaction));

}

public string ToTableRow()

{

return '\u2502' + saleDate.ToTableRow() + GoodsTransaction.ToTableRow() + String.Format("{0, 7: 0.0}\u2502{1, 14:0.0}\u2502", amount, TotalPrice) + SellerTransaction.ToTableRow();

}

}

}

## Class Balance

using System;

using System.Collections.Generic;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

namespace CourseWork.Model

{

[Serializable()]

class Balance : IObservable, ISpreadsheet

{

List<Sale> balance; //список операцій

public delegate bool Compare(Sale sale, object obj);

public delegate Sale Statistics();

protected event NotifyObserver send;

public Balance()

{

balance = new List<Sale>();

}

public event NotifyObserver Send

{

add

{

if (value != null)

{

send += value;

}

}

remove

{

if (value != null)

{

send -= value;

}

}

}

public int Length

{

get

{

return balance.Count;

}

}

public double TotalCash

{

get

{

double totalCash = 0.0;

foreach (Sale sale in balance)

{

totalCash += sale.TotalPrice;

}

return totalCash;

}

}

public void WriteToBinaryFile(string fileName)

{

Stream stream = null;

BinaryFormatter writer;

try

{

stream = File.Open(fileName, FileMode.OpenOrCreate, FileAccess.Write);

writer = new BinaryFormatter();

writer.Serialize(stream, balance);

}

finally

{

if (stream != null)

{ stream.Close(); }

}

}

public static List<Sale> ReadFromBinaryFile(string fileName)

{

List<Sale> readInformation;

Stream stream = null;

try

{

stream = File.Open(fileName, FileMode.Open);

BinaryFormatter bFormatter = new BinaryFormatter();

readInformation = (List<Sale>)bFormatter.Deserialize(stream);

}

finally

{

if (stream != null)

{

stream.Close();

}

}

return readInformation;

}

public static List<Sale> ReadFromTextFile(string fileName)

{

StreamReader reader = null;

List<Sale> readInformation = null;

Sale record;

string [] inputFields;

try

{

reader = File.OpenText(fileName);

readInformation = new List<Sale>();

while (reader.Peek() != -1) // поки не кінець файлу

{

inputFields = reader.ReadLine().Split(' ');

record = new Sale(inputFields[0], inputFields[1],

inputFields[2], inputFields[3], inputFields[4],

inputFields[5], inputFields[6], inputFields[7],

inputFields[8], inputFields[9], inputFields[10],

inputFields[11]);

readInformation.Add(record);

}

}

finally

{

if (reader != null)

{

reader.Close();

}

}

return readInformation;

}

public void WriteToTextFile(string fileName)

{

StreamWriter writer = null;

try

{

writer = new StreamWriter(fileName);

foreach (Sale element in balance)

{

writer.WriteLine(element.ToString());

}

}

finally

{

if (writer != null)

{ writer.Close(); }

}

}

public Sale this[int index]

{

get

{

if (index >= 0 && index < Length)

{ return balance[index]; }

else

{ throw new IndexOutOfRangeException("Індекс колекції за межами діапазону"); }

}

}

public void Add(Sale sale)

{

if (sale != null)

{

if (balance.Contains(sale))

return;

int i = 0;

while (i < balance.Count && balance[i].SaleDate < sale.SaleDate)

{

i++;

}

balance.Insert(i, sale);

Notify();

}

else

{

throw new ArgumentNullException();

}

}

public void Notify()

{

if (send != null)

send();

}

public Balance Search(Compare del, object obj)

{

Balance sales = new Balance();

foreach (Sale element in balance)

{

if (del(element, obj))

{

sales.Add(element);

}

}

return sales;

}

public bool CompareByName(Sale sale, object name)

{

string field = (name as string).ToUpper();

return (sale.GoodsTransaction.Name.ToUpper().Equals(field));

}

public bool CompareByPrice(Sale sale, object price)

{

return (sale.GoodsTransaction.Price.Equals(price));

}

#endregion

public Sale Max()

{

Sale max;

try

{

max = balance[0];

}

catch

{

throw;

}

foreach (Sale sale in balance)

{

if (sale.TotalPrice > max.TotalPrice)

max = sale;

}

return max;

}

public Sale Min()

{

Sale min;

try

{

min = balance[0];

}

catch

{

throw;

}

foreach (Sale sale in balance)

{

if (sale.TotalPrice < min.TotalPrice)

min = sale;

}

return min;

}

public string ToTableRow()

{

StringBuilder rows = new StringBuilder();

foreach (Sale element in balance)

{

rows.Append(element.ToTableRow());

}

return rows.ToString();

}

}

}

## Interface ITable

namespace CourseWork.Model

{

interface ISpreadsheet

{

string ToTableRow();

}

}

# Додаток 2

## Опис програми

Програма призначення для роботи з операціями продажу.

Розроблено просте інтуїтивне меню, що дозволяє робити вибір стрілками клавіш.

Програма має функції для запису в текстовий чи бінарний файл та зчитування з нього.

Розроблені методи правильної перевірки вводу нових операцій.

## Робота програми





